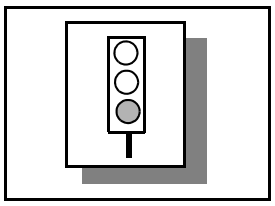
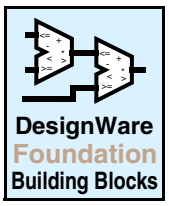




DW_arb_fcfs

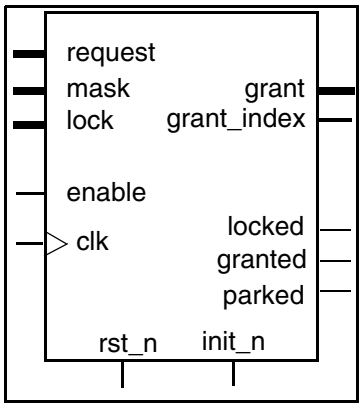
Arbiter with First-Come-First-Served Priority Scheme

Version, STAR, and myDesignWare Subscriptions: IP Directory

Features and Benefits

- Parameterizable number of clients
- Programmable mask for all clients
- Park feature - default grant when no requests are pending
- Lock feature - ability to lock the currently granted client
- Registered/unregistered outputs

Revision History



Applications

- Control application
- Networking
- Bus interfaces

Description

DW_arb_fcfs implements a parameterized, synchronous arbiter based on first-come-first-served priority scheme. In this scheme, on a cycle basis, the client that has been waiting the longest to be issued the grant, has the highest priority and the client has just been granted has the lowest priority.

Table 1-1 Pin Description

Pin Name	Width	Direction	Function
clk	1 bit	Input	Input clock
rst_n	1 bit	Input	Asynchronous reset for all registers (active low)
init_n	1 bit	Input	Synchronous reset for all registers (active low)
enable	1 bit	Input	Enables clocking (active high)
request	*n* bits	Input	Input request from clients
lock	*n* bits	Input	Active high signal to lock the grant to the current request ■ For lock(*i*) = 1, the arbiter is locked to the request (*i*) if it is currently granted. ■ For lock (*i*) = 0, the lock on the arbiter is removed.

Table 1-1 Pin Description (Continued)

Pin Name	Width	Direction	Function
mask	*n* bits	Input	Active high input to mask specific clients ■ For mask(*i*) = 1, request(*i*) is masked. ■ For mask(*i*) = 0, the mask on the request(*i*) is removed.
parked	1 bit	Output	Flag to indicate that there are no requesting clients and the grant of resources has defaulted to *park_index*
granted	1 bit	Output	Flag to indicate that arbiter has issued a grant to one of the clients
locked	1 bit	Output	Flags that the arbiter is locked by a client
grant	*n* bits	Output	Grant output
grant_index	ceil($\log_2 n$) bits	Output	Index of the requesting client that has been currently granted or the client designated by *park_index* in *park_mode*

Table 1-2 Parameter Description

Parameter	Values	Description
n	2 to 32 Default: 4	Number of arbiter clients
park_mode	0 or 1 Default: 1	■ 1: Includes logic to enable parking when no clients are requesting ■ 0: Contains no logic for parking
park_index	0 to *n*–1 Default: 0	Index of the client used for parking
output_mode	0 or 1 Default: 1	■ 1: Includes registers at the outputs (see Figure 1-1 on page 4) ■ 0: Contains no output registers (see Figure 1-2 on page 4)

Table 1-3 Synthesis Implementations

Implementation Name	Function	License Feature Required
rtl	Synthesis model	DesignWare

Table 1-4 Simulation Models

Model	Function
DW05.DW_ARB_fcfs_SIM_CFG	Design unit name for VHDL simulation
dw/dw05/DW_arb_fcfs_sim.vhd	VHDL simulation model source code
dw/sim_ver/DW_arb_fcfs.v	Verilog simulation model source code

Table 1-5 Arbiter Status Flags

Flag	Characteristic	Description
parked	If parked is active, there are no active requests at the input of the arbiter.	The parked output, active high, indicates that grant of the resources has defaulted to the client defined by *park_index* in *park_mode* = 1. In *park_mode* = 0, this flag does not exist.
granted	If granted is active, there is at least one active request at the input of the arbiter.	The granted output, active high, indicates that the grant of resources is to one of the actively requesting inputs.
locked	If locked is active, the current grant and the corresponding lock signal must be active.	The locked output, active high, indicates that the currently granted client has locked out all other clients.

If two clients assert request input in the same cycle, the DW_arb_fcfs uses the index of inputs to break the tie among the requesting clients. For example, in such cases, the client connected to the input `request[0]` has the highest priority, while the client connected to `request[`*n*`-1]` has the lowest priority.

The lock feature enables a client, despite requests from other clients, to have an exclusive grant for the duration of the corresponding lock input. After a client receives the grant, it can lock out other clients from the arbitration process by setting the corresponding lock input.

The park feature allows the resources to be granted to a designated client defined by the *park_index* parameter when there are no active requests pending. The *park_mode* and *lock_mode* parameters enable/disable these features.

By setting the desired bits of the `mask` input, the corresponding clients can be masked off from consideration for arbitration. The mask on a client remains active until the corresponding mask input for the client is reset.

All the input requests from the arbiter clients are assumed to be synchronous to the arbiter clock signal `clk`.

The arbiter provides flags: `locked`, `granted` and `parked`, to indicate the status of the arbiter. Table 1-5 on page 3 shows a detailed description of all the flags of the arbiter.

Figure 1-1 Block Diagram of DW_arb_fcfs Arbiter, output_mode = 1

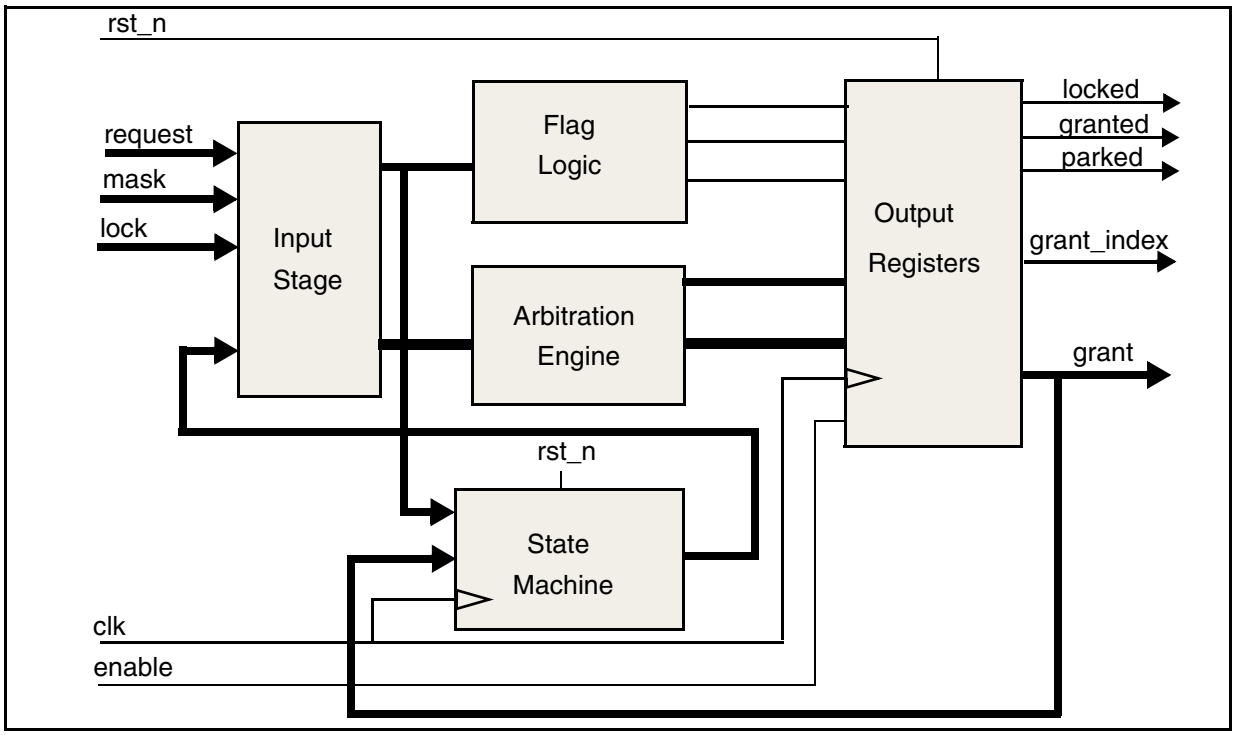
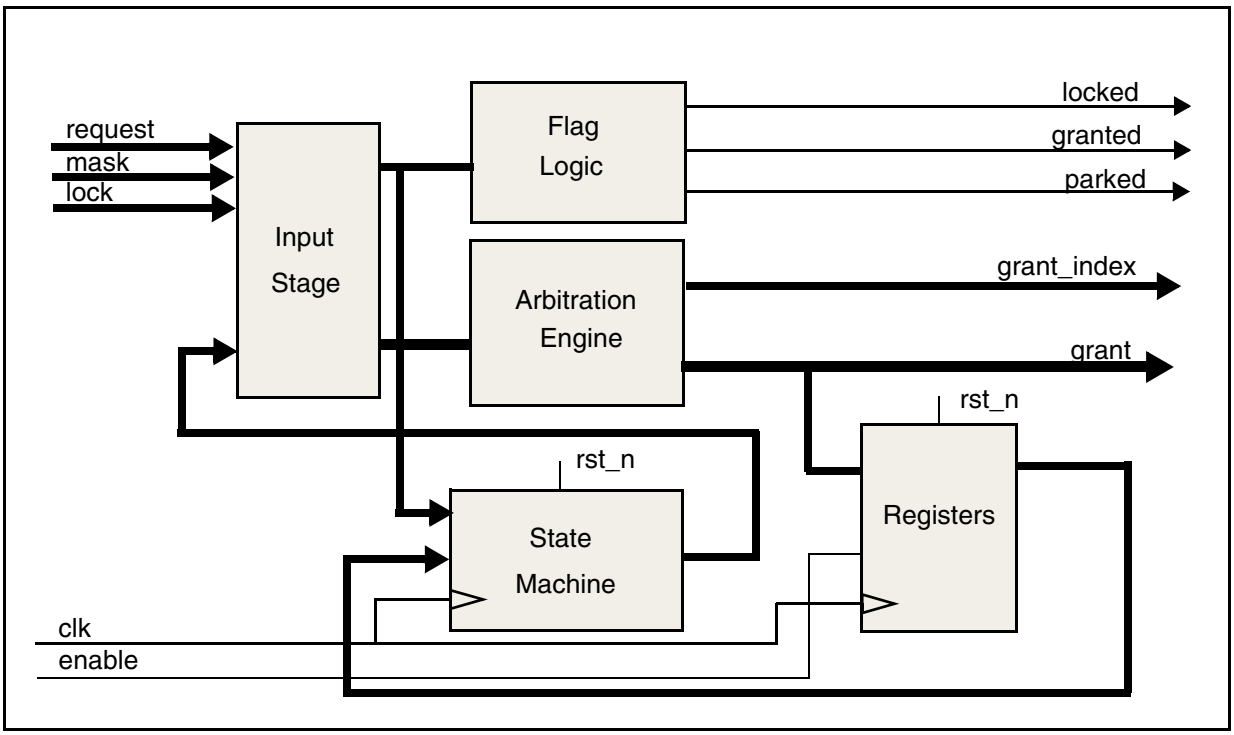


Figure 1-2 Block Diagram of DW_arb_fcfs Arbiter output_mode = 0



Functional Description

The DW_arb_fcfs internally computes and maintains the priorities of all the clients based on the current grant and actively requesting clients. With no active requests, the priorities of all the clients are set to the lowest possible priority for the number of clients connected to the arbiter. While the priority of the currently granted client is set to the lowest priority, the priorities of all the requesting clients not yet been granted, are increased by one. Since the priorities are updated each cycle, the grant is issued to one of the actively requesting clients on a first-come-first-served basis every cycle. As mentioned earlier, the index of the clients to the arbiter is used to any potential deadlock in case the internal priorities of two or more clients are the same.

The internal priorities are updated based on the current state of the arbiter. The criteria used to update the priorities are as follows:

- The non-requesting inputs have their internal priorities set to lowest value.
- The internal priorities of actively requesting inputs increased by one each cycle until granted.
- The internal priority of the currently granted client is set to the lowest value in the next cycle.
- In the lock state the internal priorities of actively requesting inputs are held at levels they were prior to entering the lock state. But if any of the inputs de-asserts the request in the lock state, its internal priority is set to the lowest level.

The mask, park and lock features add flexibility to the arbiter. The parking of grant to a designated client saves an arbitration cycle and the parked client can lock the grant without issuing a request to the arbiter.

Any client can be masked off by setting the corresponding mask bit. By doing so it will not be considered for the arbitration. If mask bits are set and none of the non-masked clients are actively requesting, the arbiter will be parked to the designated client defined by *park_index*. In the non-locked state of the arbiter, setting the mask bit of the currently granted client effectively invalidates the request from the client. In the following cycle, the current grant is de-asserted, and based on the current unmasked requests from other clients, a new client is generated. However, when a client has locked the arbiter, setting the mask bit of any client has no effect on the current grant.

Suppressing Warning Messages During Verilog Simulation

The Verilog simulation model includes macros that allow you to suppress warning messages during simulation.

To suppress all warning messages for all DWBB components, define the DW_SUPPRESS_WARN macro in either of the following ways:

- Specify the Verilog preprocessing macro in Verilog code:

  ```
  `define DW_SUPPRESS_WARN
  ```

- Or, include a command line option to the simulator, such as:

  `+define+DW_SUPPRESS_WARN` (which is used for the Synopsys VCS simulator)

The warning messages for this model include the following:

- If values other than 1 or 0 are present on a clock port, the following message is displayed:

  ```
  WARNING: <instance_path>.<clock_name>_monitor:
      at time = <timestamp>, Detected unknown value, x, on <clock_name> input.
  ```

  To suppress only this warning message for all DWBB components, use the following macro:

  - Define the DW_DISABLE_CLK_MONITOR macro. You can define this macro in the following ways:
    - Specify the Verilog preprocessing macro in Verilog code:

      ```
      `define DW_DISABLE_CLK_MONITOR
      ```

    - Or, include a command line option to the simulator, such as:

      `+define+DW_DISABLE_CLK_MONITOR` (which is used for the Synopsys VCS simulator)

  This message is also suppressed using the DW_SUPPRESS_WARN macro explained earlier.

Timing Waveforms

The following figures shows timing diagrams for various conditions:

Figure 1-3 Waveform 1

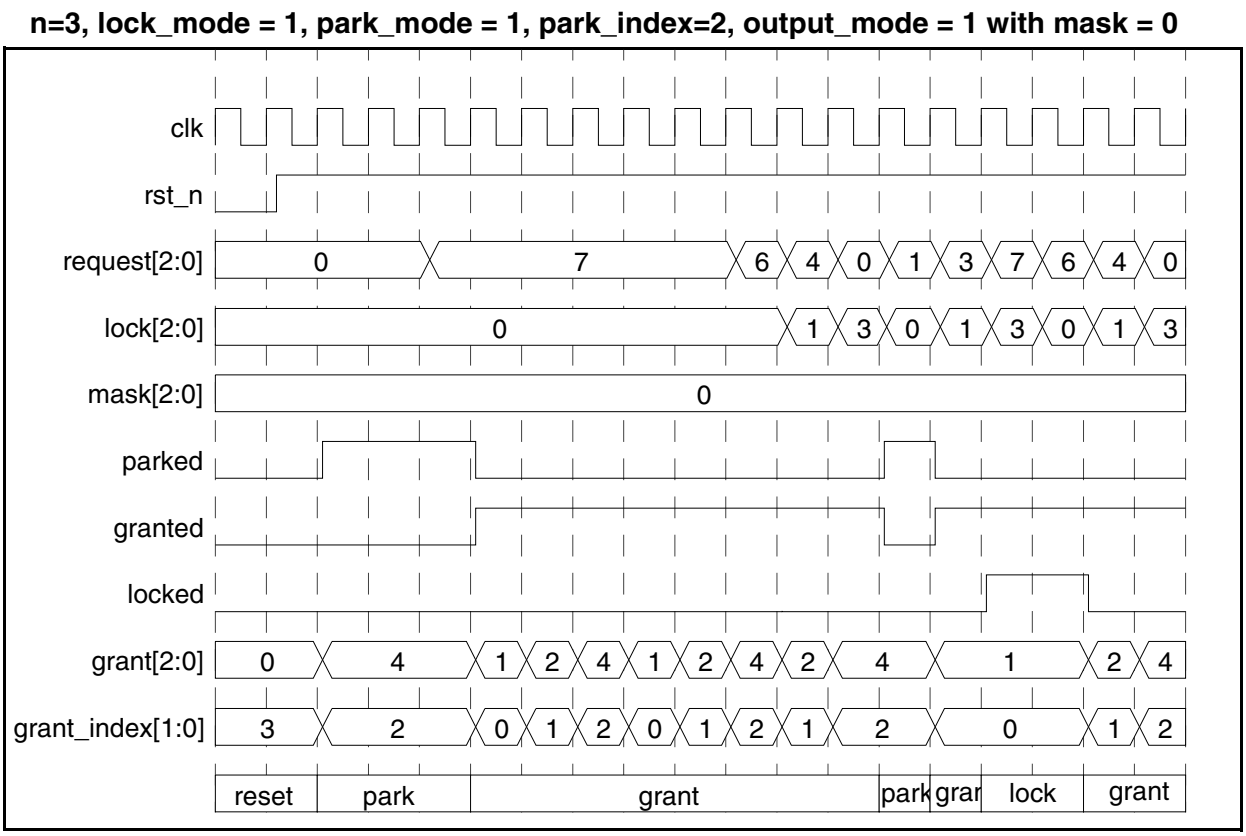


Figure 1-4 Waveform 2

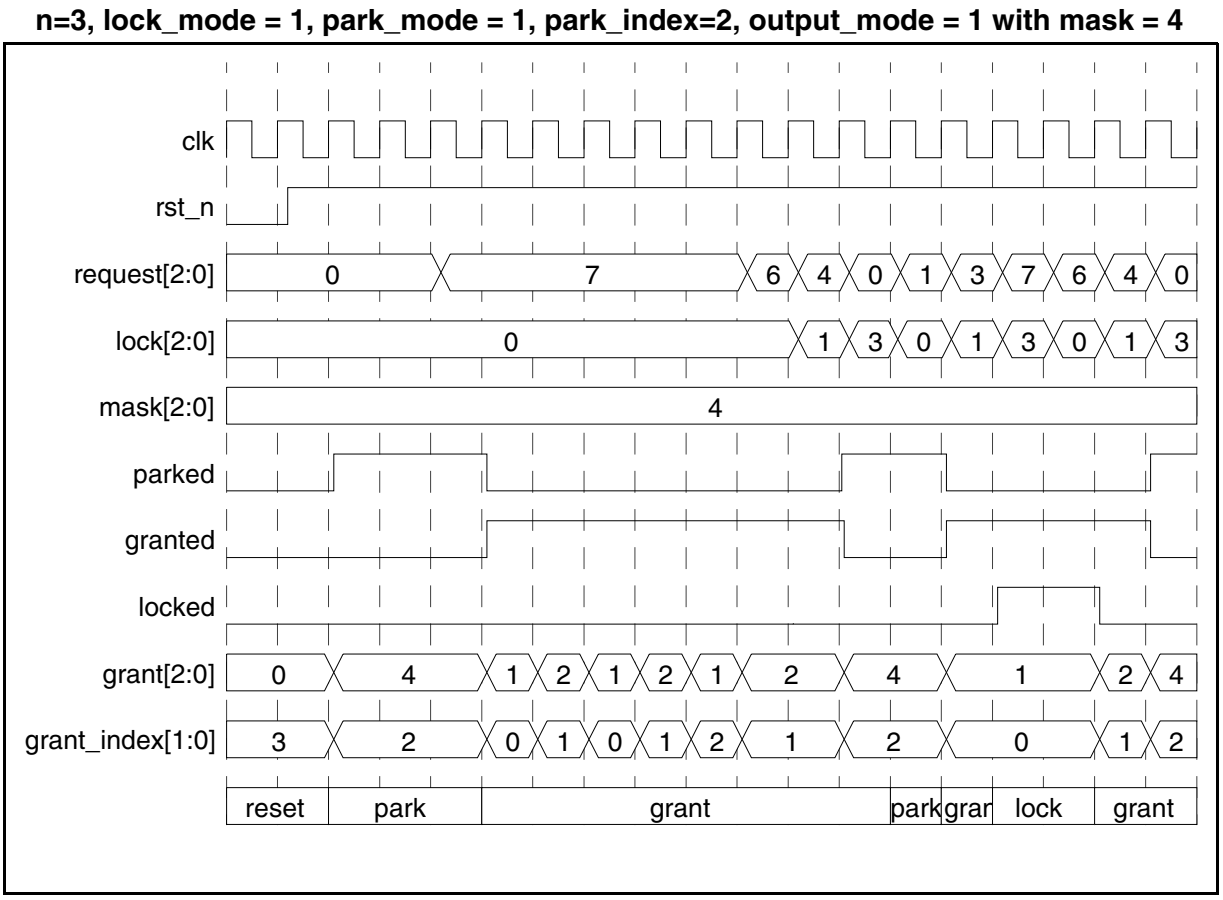
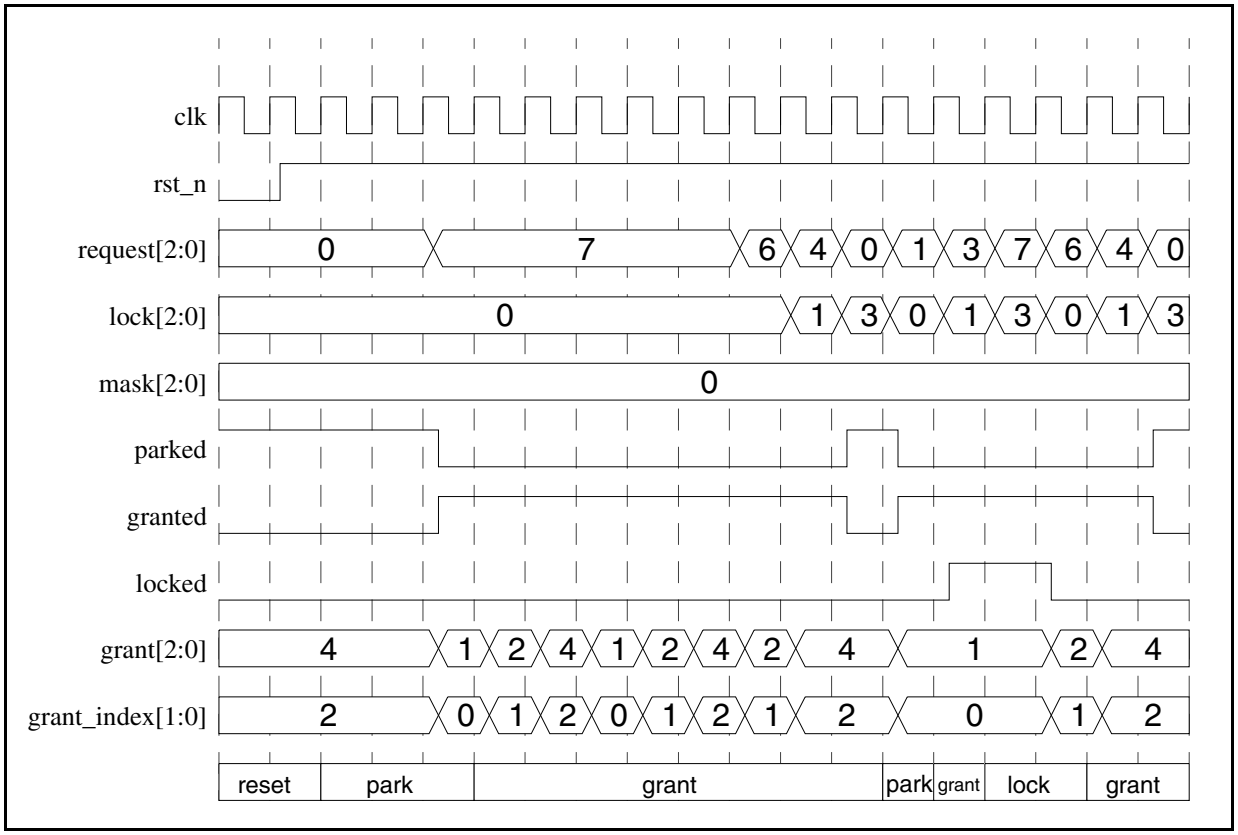


Figure 1-5 Waveform 3

n=3, lock_mode = 1, park_mode = 1, park_index=2, output_mode = 0 with mask = 0



Related Topics

- Application Specific – Control Logic Overview
- DesignWare Building Block IP User Guide

HDL Usage Through Component Instantiation - VHDL

```
library IEEE,DWARE;
use IEEE.std_logic_1164.all;
use DWARE.DWpackages.all;
use DWARE.dw_foundation_comp.all;

entity DW_arb_fcfs_inst is
      generic (
        inst_n : NATURAL := 4;
        inst_park_mode : NATURAL := 1;
        inst_park_index : NATURAL := 0;
        inst_output_mode : NATURAL := 1
        );
      port (
        inst_clk : in std_logic;
        inst_rst_n : in std_logic;
        inst_init_n : in std_logic;
        inst_enable : in std_logic;
        inst_request : in std_logic_vector(inst_n-1 downto 0);
        inst_lock : in std_logic_vector(inst_n-1 downto 0);
        inst_mask : in std_logic_vector(inst_n-1 downto 0);
        parked_inst : out std_logic;
        granted_inst : out std_logic;
        locked_inst : out std_logic;
        grant_inst : out std_logic_vector(inst_n-1 downto 0);
        grant_index_inst : out std_logic_vector(bit_width(inst_n)-1 downto 0)
        );
    end DW_arb_fcfs_inst;


architecture inst of DW_arb_fcfs_inst is

begin

    -- Instance of DW_arb_fcfs
    U1 : DW_arb_fcfs
    generic map (
          n => inst_n,
          park_mode => inst_park_mode,
          park_index => inst_park_index,
          output_mode => inst_output_mode
          )
    port map (
          clk => inst_clk,
          rst_n => inst_rst_n,
          init_n => inst_init_n,
          enable => inst_enable,
          request => inst_request,
```

```
            lock => inst_lock,
            mask => inst_mask,
            parked => parked_inst,
            granted => granted_inst,
            locked => locked_inst,
            grant => grant_inst,
            grant_index => grant_index_inst
            );


  end inst;

  -- pragma translate_off
  configuration DW_arb_fcfs_inst_cfg_inst of DW_arb_fcfs_inst is
    for inst
    end for; -- inst
  end DW_arb_fcfs_inst_cfg_inst;
  -- pragma translate_on
```

HDL Usage Through Component Instantiation - Verilog

```
module DW_arb_fcfs_inst( inst_clk, inst_rst_n, inst_init_n, inst_enable, inst_request,
          inst_lock, inst_mask, parked_inst, granted_inst, locked_inst,
          grant_inst, grant_index_inst );

parameter inst_n = 4;
parameter inst_park_mode = 1;
parameter inst_park_index = 0;
parameter inst_output_mode = 1;

`define bit_width_n 2// bit_width_n is set to ceil(log2(n))

input inst_clk;
input inst_rst_n;
input inst_init_n;
input inst_enable;
input [inst_n-1 : 0] inst_request;
input [inst_n-1 : 0] inst_lock;
input [inst_n-1 : 0] inst_mask;
output parked_inst;
output granted_inst;
output locked_inst;
output [inst_n-1 : 0] grant_inst;
output [`bit_width_n-1 : 0] grant_index_inst;

    // Instance of DW_arb_fcfs
    DW_arb_fcfs #(inst_n, inst_park_mode, inst_park_index, inst_output_mode) U1 (
                .clk(inst_clk),
                .rst_n(inst_rst_n),
                .init_n(inst_init_n),
                .enable(inst_enable),
                .request(inst_request),
                .lock(inst_lock),
                .mask(inst_mask),
                .parked(parked_inst),
                .granted(granted_inst),
                .locked(locked_inst),
                .grant(grant_inst),
                .grant_index(grant_index_inst) );

endmodule
```

Revision History

For notes about this release, see the *DesignWare Building Block IP Release Notes*.

For lists of both known and fixed issues for this component, refer to the STAR report.

For a version of this datasheet with visible change bars, click here.

Date	Release	Updates
July 2020	DWBB_201912.5	■ Adjusted content and title of "Suppressing Warning Messages During Verilog Simulation" on page 6 and added the DW_SUPPRESS_WARN macro
October 2019	DWBB_201903.5	■ Added the "Disabling Clock Monitor Messages" section
March 2019	DWBB_201903.0	■ Removed minPower designation from this datasheet ■ Added this Revision History table and the document links on this page

Copyright Notice and Proprietary Information

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at https://www.synopsys.com/company/legal/trademarks-brands.html.

All other product or company names may be trademarks of their respective owners.

Free and Open-Source Software Licensing Notices

If applicable, Free and Open-Source Software (FOSS) licensing notices are available in the product installation.

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

Synopsys, Inc.
www.synopsys.com